

Tarea Complementaria 3

```
package tarea.complementaria.pkg3.random.access.file.con.indice;

import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Scanner;

/**
 * Tarea complementaria 3:
 * Uso de RandomAccessFile con un fichero de índice auxiliar.
 *
 * personas.dat -> fichero de DATOS con registros FIJOS de 48 bytes:
 *           id (int, 4 B) + nombre (20 chars, 40 B) + edad (int, 4 B)
 *
 * personas.idx -> fichero de ÍNDICE con entradas de 12 bytes:
 *           id (int, 4 B) + offset (long, 8 B)
 *
 * El flujo del programa es:
 * 1) Nos aseguramos de que personas.dat tenga algunos registros de ejemplo.
 * 2) Recorremos personas.dat y reconstruimos personas.idx desde cero.
 * 3) Pedimos un id por teclado y lo buscamos en personas.idx.
 * 4) Si lo encontramos, usamos el offset para saltar DIRECTAMENTE
 *    al registro correspondiente en personas.dat y lo mostramos.
 */

public class TareaComplementaria3RandomAccessFileConIndice {

    // ----- Formato del registro en personas.dat -----

    // Número de caracteres fijos para el campo nombre (20 chars -> 40 bytes)
    static final int NCHARS = 20;

    // Tamaño total del registro en bytes:
    //   id (4 B) + nombre (20 chars * 2 B) + edad (4 B) = 48 bytes
    static final int BYTES_REG = 4 + 2 * NCHARS + 4;

    // ----- Formato de una entrada en personas.idx -----

    // Tamaño de cada entrada del índice:
    //   id (4 B) + offset (8 B) = 12 bytes
    static final int BYTES_IDX = 4 + 8;

    public static void main(String[] args) throws IOException {

        // try-with-resources:
        //   - sc: lectura de datos desde consola
        //   - rafData: acceso aleatorio al fichero de DATOS (personas.dat)
        //   - rafIdx: acceso aleatorio al fichero de ÍNDICE (personas.idx)
        try (Scanner sc = new Scanner(System.in);
             RandomAccessFile rafData = new RandomAccessFile("personas.dat", "rw");
             RandomAccessFile rafIdx = new RandomAccessFile("personas.idx", "rw")) {
```

```

    // 1) Si personas.dat está vacío (length == 0), generamos 4 registros de
ejemplo.

    // Esto sólo ocurre la primera vez que ejecutamos el programa.
    if (rafData.length() == 0) {
        // Cada llamada escribe un registro completo (id, nombre, edad)
        // en la POSICIÓN ACTUAL del puntero de datos.
        escribirRegistro(rafData, 10, "Ana", 30); // reg 0 -> offset 0
        escribirRegistro(rafData, 20, "Luis", 25); // reg 1 -> offset 48
        escribirRegistro(rafData, 30, "Maria", 40); // reg 2 -> offset 96
        escribirRegistro(rafData, 40, "Pepe", 22); // reg 3 -> offset 144
    }

    // 2) Reconstruimos el índice desde cero.
    // Recorremos TODOS los registros de personas.dat y por cada uno
escribimos
    // en personas.idx el par (id, offsetInicioRegistro).
    construirIndice(rafData, rafIdx);

    // 3) Pedimos al usuario la clave de búsqueda (id)
    System.out.print("Introduce ID a consultar: ");
    int idBuscado = sc.nextInt();

    // 4) Buscamos el offset correspondiente en el fichero de índice.
    // Si existe una entrada con ese id, nos devolverá el byte
    // exacto donde empieza el registro en personas.dat.
    long offset = buscarOffsetEnIndice(rafIdx, idBuscado);

    if (offset == -1) {
        // Caso en el que no hay ninguna entrada en el índice con ese id.
        System.out.println("ID no encontrado en el índice.");
        return; // Terminamos el programa aquí.
    }

    // 5) Si lo hemos encontrado, saltamos DIRECTAMENTE al registro en
personas.dat
    // usando seek(offset). Así evitamos recorrer todo el fichero de datos.
    rafData.seek(offset); // El puntero queda justo al inicio
del registro

    // Leemos el registro en el mismo ORDEN en que se escribió:
    int id = rafData.readInt(); // id (4 bytes)
    String nombre = leerNombreFijo(rafData, NCHARS); // nombre (20 chars -> 40
B)
    int edad = rafData.readInt(); // edad (4 bytes)

    // 6) Mostramos por pantalla el registro recuperado usando el índice
    System.out.println("Registro encontrado -> id=" + id +
                       ", nombre=" + nombre +
                       ", edad=" + edad);
}

// -----
// MÉTODOS AUXILIARES

```

```

// -----



/** 
 * Recorre personas.dat y construye personas.idx desde cero.
 *
 * Por cada registro i en personas.dat:
 *   - Calcula el offset de inicio del registro: i * BYTES_REG
 *   - Se posiciona en ese offset y lee sólo el id
 *   - Escribe en personas.idx:
 *     id (int, 4 bytes) + offset (long, 8 bytes)
 */
static void construirIndice(RandomAccessFile rafData,
                            RandomAccessFile rafIdx) throws IOException {

    // Dejamos el índice vacío antes de reconstruirlo:
    // si ya existía un personas.idx previo, lo sobrescribimos desde cero.
    rafIdx.setLength(0);

    // Número de registros que hay en el fichero de DATOS:
    // tamaño_total_bytes / tamaño_de_un_registro
    long numRegs = rafData.length() / BYTES_REG;

    // Recorremos cada registro de datos
    for (int i = 0; i < numRegs; i++) {

        // Offset (en bytes) donde empieza el registro i:
        // regOffset = i * 48 (0, 48, 96, 144, ...)
        long regOffset = i * (long) BYTES_REG;

        // Nos colocamos al inicio del registro i en personas.dat
        rafData.seek(regOffset);

        // Leemos SÓLO el id del registro (4 bytes).
        // No necesitamos leer nombre ni edad para el índice.
        int id = rafData.readInt();

        // Nos colocamos al final de personas.idx (ya está listo para escribir)
        // y añadimos una nueva entrada con:
        //   - id    -> clave de búsqueda
        //   - offset-> byte donde empieza el registro en personas.dat
        rafIdx.writeInt(id);           // 4 bytes
        rafIdx.writeLong(regOffset); // 8 bytes
    }
}

/** 
 * Busca linealmente en personas.idx una entrada con el id indicado.
 *
 * Formato de cada entrada en personas.idx:
 *   - id (int, 4 bytes)
 *   - offset (long, 8 bytes)
 *
 * Recorrido:
 *   1) Posicionamos el puntero al inicio (seek(0)).

```

```

*   2) Mientras queden bytes por leer:
*       - Leemos un id (4 B).
*       - Leemos su offset (8 B).
*       - Si el id coincide, devolvemos el offset.
*   3) Si llegamos al final del fichero sin encontrarlo, devolvemos -1.
*/
static long buscarOffsetEnIndice(RandomAccessFile rafIdx,
                                  int idBuscado) throws IOException {

    // Empezamos siempre desde el principio del archivo de índice
    rafIdx.seek(0);

    // Bucle de lectura secuencial de entradas (id, offset)
    while (rafIdx.getFilePointer() < rafIdx.length()) {

        // Leemos el id almacenado en esta entrada (4 bytes)
        int id = rafIdx.readInt();

        // Leemos el offset asociado a ese id (8 bytes)
        long off = rafIdx.readLong();

        // Si coincide con el id que queremos consultar, devolvemos el offset
        if (id == idBuscado) {
            return off;
        }
    }

    // Si hemos salido del bucle, no se ha encontrado ninguna entrada con ese id
    return -1;
}

/**
 * Escribe un registro COMPLETO (id, nombre fijo, edad) en la POSICIÓN ACTUAL
 * del puntero de datos de personas.dat.
 *
 * IMPORTANTE: este método asume que el puntero ya está bien colocado
 * (por ejemplo, al final del archivo o en el inicio de un registro concreto).
 */
static void escribirRegistro(RandomAccessFile raf,
                            int id,
                            String nombre,
                            int edad) throws IOException {

    // 1) Escribimos el id como int (4 bytes)
    raf.writeInt(id);

    // 2) Escribimos el nombre como campo de tamaño FIJO:
    //     exactamente NCHARS caracteres -> 40 bytes.
    //     Si el String es más corto, se rellena con espacios;
    //     si es más largo, se recorta.
    escribirNombreFijo(raf, nombre, NCHARS);

    // 3) Escribimos la edad como int (4 bytes)
    raf.writeInt(edad);
}

```

```

}

/** 
 * Escribe una cadena de tamaño fijo 'len' en el fichero.
 *
 * - Si s tiene menos de len caracteres -> se rellenan espacios al final.
 * - Si s tiene más de len caracteres -> se recorta.
 *
 * Cada carácter se escribe con writeChar() -> 2 bytes por char.
 */
static void escribirNombreFijo(RandomAccessFile raf,
                                String s,
                                int len) throws IOException {

    // "%-20s" -> alinea el texto a la IZQUIERDA y rellena con espacios
    //           hasta que la longitud sea 20 chars.
    // Aquí usamos "%-len s" para que sea genérico.
    String fijo = String.format("%-" + len + "s", s);

    // Recorremos los len caracteres del String formateado
    for (int i = 0; i < len; i++) {
        // writeChar escribe internamente 2 bytes (UTF-16)
        raf.writeChar(fijo.charAt(i));
    }
}

/** 
 * Lee exactamente 'len' caracteres del fichero y los devuelve como String,
 * eliminando espacios de relleno al principio y al final.
 *
 * Se corresponde con lo que escribió escribirNombreFijo():
 *   - len llamadas a writeChar() -> len llamadas a readChar()
 */
static String leerNombreFijo(RandomAccessFile raf,
                             int len) throws IOException {

    // Buffer donde vamos acumulando los caracteres leídos
    StringBuilder sb = new StringBuilder(len);

    // Leemos len caracteres; cada readChar consume 2 bytes del fichero
    for (int i = 0; i < len; i++) {
        sb.append(raf.readChar());
    }

    // trim():
    //   - elimina espacios al principio y al final
    //   - nos devuelve el nombre tal y como lo escribimos originalmente
    //     (sin relleno extra).
    return sb.toString().trim();
}
}

```